PROGRAMMABLE CONTROLLER, AND

CONTROL-PROGRAM-DEVELOPMENT SUPPORTING APPARATUS

FIELD OF THE INVENTION

5   The present invention relates to a programmable
controller that performs sequential processing in
accordance with execution codes generated by compiling a
control program such as an instruction list or ladder diagram
and a control-program-development supporting apparatus for
10   compiling and debugging the control program.

BACKGROUND OF THE INVENTION

Production equipments and production methods have been
rapidly changed in industrial fields, particularly in
15   production fields independently of the scale of a field and
the range of electrical-system controllers for equipments
and systems for production has been increased year by year.
Therefore, it is important how to communicate with a
mechanical system, rationally improve the cost performance,
20   and realize a high-speed easy-to-handle user interface under
the above environment.

Conventionally, programmable controllers (PC) and
development environment for controlling the controllers
have been developed that realize the above-mentioned
25   conditions. A programmable controller is also referred to

1

as a programmable logic controller (PLC) or sequencer. The programmable controller generally operates in accordance with the description of a special control program for sequential processing.

5   The sequential processing has a basic processing flow of receiving a signal from an input device such as a sensor or switch and transmitting a control signal to an output device such as an indicator or electromagnetic relay in accordance with a state of the received signal or a specific

10   order. That is, a control program is a program in which the flow of the above sequential processing is described.

In general, the control program is developed by a universal computer such as a personal computer or program loader   (hereafter   referred   to   as

15   control-program-development supporting apparatus) and transferred to a programmable controller through a communication interface.

The ladder diagram system, function-block diagram system, structured text system, and SFC (Sequential Function

20   Chart) system are typically known among languages for describing a control program. Particularly, the ladder diagram system is a system in which a circuit diagram is drawn like a ladder, two vertical lines at the both ends of the ladder symbolically show motive power, and the flow

25   of the motive power of a relay contact circuit is shown on

2

a line horizontally connected between the two lines, which is a language most widely used.

The ladder diagram system first expresses the flow of sequential processing in accordance with the arrangement of graphic symbols which can be intuitively understood by a person as described above and then converts the arrangement of the graphic symbols into an instruction list shown by mnemonic codes which can be interpreted by a programmable controller by a converter such as a control-program-development supporting apparatus.

Therefore, the programmable controller may use a configuration for receiving the instruction list and executing sequential processing in accordance with instructions shown by the instruction list. In this case, the programmable controller sequentially executes a control program by interpreting the received instruction list one line by one line, that is, converting the list into directly-executable codes by a microprocessor mounted on the programmable controller. The programmable controller is hereafter referred to as interpreter-type programmable controller.

Moreover, a compiling-type programmable controller is practically used as another type of programmable controller that receives directly-executable codes from a control-program-development supporting apparatus by a

3

microprocessor mounted on the programmable controller and executes the codes. In the case of the configuration using the compiling-type programmable controller, a control-program-development supporting apparatus is provided with a compiler, which converts a ladder diagram directly or temporarily into an instruction list by the compiler and then, compiles the list into directly-executable codes by the microprocessor and transfers the execution codes to the programmable controller.

However, because the above control program depends on the bender or type of a programmable controller, it is impossible to execute the same sequential processing between different types of programmable controllers by using control programs having the same source code. Therefore, it is necessary to develop a new control program or transplant a new model or different model to a programmable controller and thus, it is difficult to effectively use past properties of a control program.

Therefore, a method is proposed which obtains execution codes by describing a control program with not a conventional sequential-processing language such as a ladder diagram or instruction list but an advanced programming language used for a personal computer and compiling the advanced programming language with a compiler.

4

For example, according to the advanced-language sequence instruction program generator and universal sequencer disclosed in Japanese Patent Laid-Open No. HEI 7-295612, the advanced-language sequence instruction

5　program generator translates a sequence instruction language sequence instruction program into an advanced-language sequence instruction program with a translation program and compiles the advanced-language sequence instruction program to generate an execution-type

10　sequence instruction program. Or, the universal sequencer executes the advanced-language sequence instruction program while sequentially interpreting the program. Thereby, it is possible to execute sequential control by the same control program independently of the bender or type

15　of a sequencer.

Moreover, the following two methods are known in order to update the contents of a control program stored in a program controller. One of them is a method in which a control-program-development supporting apparatus

20　transfers a control program to a programmable controller currently stopped and the programmable controller starts execution processing after reception of the control program is completed.

The other of them is a method (hereinafter, referred

25　to as under-RUN writing method) in which a programmable

controller rewrites the contents of a control program currently executed to contents of a new control program while continuing the execution state. A programmable controller normally operates by repeatedly interleaving and executing

5   a loop of a binary-code string corresponding to each instruction to be executed one-to-one. This method is realized by first reading a new control program into an idle memory space and when reading of the program is completed, stopping the execution of an original control program and

10   starting the execution of the new control program at the start or end timing of the loop.

     Moreover, a so-called software programmable controller is practically used in which development and debugging of a control program realized by the above

15   control-program-development supporting apparatus and sequential processing of a programmable controller are realized by one system. The configuration of the software programmable controller is the same as that of a universal computer, in which sequential processing to external input

20   and output devices is realized by software.

     However, the above-described programmable controller and control-program-development supporting apparatus have the following problems.

     Firstly, there is a problem that, because a

25   microprocessor mounted on a universal computer (hereafter

6

referred to as universal microprocessor) such as a personal computer operates with an advanced function at a high speed and can be inexpensively available in recent years, the advantage of developing a microprocessor mounted on a conventional programmable controller with an ASIC (Application Specific IC) is slowly decreased.

Particularly, a universal microprocessor is provided with primary and secondary caches and constructed in accordance with the acceleration mounting arts such as pipeline processing, super-scaler processing and out-of-order processing and moreover, execution codes directly executable by the universal microprocessor are also generated by a compiler for converting the execution codes into execution codes forming an optimum code system by fully using the acceleration mounting arts.

Because most compilers of the above universal computer compile a program described with an advanced programming language easily understandable by a user, it is very advantageous to mount a universal processor on a programmable control together with the above compilers and make the processor execute sequential processing. In the case of a universal computer, however, the start point of program development is set to description of an advanced programming language. Therefore, a problem occurs that it is impossible to make efficient use of a control program such as a ladder

diagram or instruction list, that is, the past property of sequential processing.

Secondly, because a control program tends to be lengthened due to diversification of input and output devices and complication of sequential processing, there is a problem that a lot of time is necessary for development, compiling, interpreting, and debugging of the control program.

Thirdly, because a compiling-type programmable controller receives an execution code and directly starts sequential processing, there is a problem that it is difficult to estimate an actual processing time corresponding to each instruction unit or processing-block unit of an original control program.

Fourthly, though the "advanced-language sequence instruction program generator" disclosed in the above Japanese Patent Laid-Open No. HEI 7-295612 generates an object file by converting a source file described with the control program of an instruction list or the like into the source file of an advanced programming language such as C language, if only the source file of the advanced programming language is corrected, there is a problem that two source files showing the same sequential processing are not matched each other because the source file of an original control program is not changed due to the above correction.

Particularly, in the case of a

control-program-development supporting apparatus capable of developing a control program with an advanced programming language, to perform step execution by using a debugging tool, for example, step execution is possible for one line
5 of the advanced programming language but it is impossible to perform step execution every line for a language configuration such as an instruction list before converted into an advanced programming language.

Moreover, in general, at a job site where a programmable
10 controller works, a control-program-development supporting apparatus capable of directly operating an advanced programming language is not set in many cases. Therefore, when a trouble occurs in sequential processing or a control program must be slightly corrected, debugging may be
15 performed by using a simple control-program-development supporting apparatus capable of using only a sequential-processing language such as an instruction list. That is, in this case, the above problem becomes remarkable. To correspond to the above problem, it is necessary to prepare
20 a control-program-development supporting apparatus capable of directly operating an advanced programming language in every job site but the cost for setting the system increases.

Fifthly, to realize the above-described under-RUN writing method, there is a problem that a memory having the
25 same size as a memory in which a control program currently

9

executed is read must be separately prepared. Because a memory is generally expensive among hardware parts, a maker or user must pay much investment in order to realize the under-RUN writing method.

5   Sixthly,   the   above-described   compiling-type control-program-development supporting apparatus has a problem that it is generally impossible to restore a control program such as an original instruction list from finally generated execution codes. Therefore, when a programmable

10   controller and a control-program-development supporting apparatus are set in housings separate from each other and they are separated from each other off-line, an original control program and an execution code actually executed are stored in places different from each other.

15   Particularly,   in   the   case   of   a   compiling-type control-program-development supporting apparatus, if the source file of a control program disappears at the control-program-development supporting apparatus side, it is impossible to restore the control program such as an

20   instruction list and debugging becomes impossible. In this case, a control program must be rewritten from the beginning.

Moreover, even if the source file of a control program does   not   disappear,   it   is   necessary   to   access   a control-program-development supporting apparatus storing

25   the source file of an original control program and fetch

the file in order to change the processing contents of a programmable controller currently operated at a job site.

Seventhly, the above programmable controller or control-program-development supporting apparatus, independently of the interpreter type or compiling type, has a problem that the procedure for interpreting processing contents is not always efficient though a part for sequentially interpreting processing contents of a control program described by a user is present. The part for performing sequential interpretation (hereafter referred to as instruction interpreting unit) serves as a part for reading each data value constituting a control program when the program is executed and interpreting the execution of an instruction designated by the data value in the case of an interpreter-type programmable controller and serves as a part for sequentially reading and interpreting a control program and converting it into a proper output format in the case of a compiling-type control-program-development supporting apparatus.

Instruction interpreting unit is generally described with a structure of comparing input data with a pattern of the interpretation side and when the data and the pattern are matched each other, executing a previously-stored operation (action). Therefore, pattern matching of the instruction interpreting unit requires a lot of time when

the operation is executed in the case of the interpreter type and when the operation is compiled in the case of the compiling type. Thus, both types are factors for impeding the productivity.

5

SUMMARY OF THE INVENTION

It is the first object of the present invention to provide a programmable controller which mounts a universal microprocessor having advantages that higher-speed sequential processing can be realized and the microprocessor can be inexpensively obtained and a control-program-development supporting apparatus for generating an execution code directly executable by a universal microprocessor from a control program such as an instruction list.

It is the second object of the present invention to provide a control-program-development supporting apparatus or programmable controller capable of minimizing a part for performing recompiling by dividing a control program such as an instruction list at a predetermined position and generating a file every divided block.

It is the third object of the present invention to provide a control-program-development supporting apparatus or programmable controller capable of estimating the execution of a control program by preparing a sample program

12

whose processing time is already known.

It is the fourth object of the present invention to provide a control-program-development supporting apparatus or programmable controller capable of debugging the source file of an advanced programming language intermediately generated and the source file of an actual control program while matching the both source files.

It is the fifth object of the present invention to provide a programmable controller for automatically replacing an execution code currently executed with a new execution code in accordance with the difference data in which only modified portions of a control program are described.

It is the sixth object of the present invention to provide a control-program-development supporting apparatus capable of unifying a control program and an execution code by compressing the source file of the control program and inserting the source file into the execution code and a programmable controller.

It is the seventh object of the present invention to provide a control-program-development supporting apparatus capable of shortening the time necessary for sequential instruction interpretation such as compiling or interpreting by preferentially pattern-matching instructions frequently used in a control program with an

13

instruction interpreting section and a programmable controller.

The programmable controller according to one aspect of the present invention comprises a storing unit which stores the execution code and a universal microprocessor which mounts an acceleration mounting unit such as a pipeline logic or cache and which is directly executed by the execution code.

According to the above invention, because a universal microprocessor which mounts acceleration mounting unit is used and a received execution code is optimized so that the code is executed by the acceleration mounting unit of the universal microprocessor, it is possible to execute a sequential operation through high-speed processing such as pipeline super-scaler, or out-of-order.

The control-program-development supporting apparatus according to one aspect of the present invention comprises a compiler which compiles the control program into execution codes directly executable by a universal microprocessor which mounts acceleration mounting unit such as a pipeline logic or cache.

According to the above invention, a control program described with a sequential-control language is compiled into execution codes executable by a universal microprocessor that mounts acceleration mounting unit.

14

Therefore, it is possible to execute a sequential operation through high-speed processing such as pipeline, super-scaler, or out-of-order by transmitting the execution codes to a programmable controller provided with the universal microprocessor and moreover, obtain execution codes corresponding to various universal microprocessors.

The control-program-development supporting apparatus according to another aspect of the present invention comprises a control-program dividing unit which divides the control program into a plurality of blocks, a compiler which compiles all or some of the blocks into execution codes directly executable by a programmable controller, and a communication interface which transmits the execution codes to the programmable controller.

According to the above invention, because it is possible to obtain final execution codes by dividing a control program into a plurality of blocks and compiling the control program every divided blocks, it is possible to generate execution codes by recompiling only portions to be modified in the control program even if the control program is a long program.

The control-program-development supporting apparatus according to still another aspect of the present invention comprises a control-program dividing unit which divides the control program into a plurality of blocks, a

15

control-program converting unit which converts all or some of the blocks into an advanced-language control program described with a universal-computer-readable advance language every blocks concerned, and a compiler which compiles the whole of a part of a corresponding advanced language into execution codes directly executable by a programmable controller every blocks concerned.

According to the above invention, a control program is divided into a plurality of blocks and each of the divided blocks is converted into a universal-computer-readable advanced language. Moreover, it is possible to compile a program described with a universal-computer-readable advanced language corresponding to each block and obtain final execution codes. Therefore, even if a long control program is used, it is possible to generate execution codes by recompiling only a portion to be modified in the program and use a compiler for a conventional universal-computer-readable advanced language.

Further, it is preferable that the programmable controller is provided with a universal microprocessor that mounts acceleration mounting unit such as a pipeline logic and a cache. Accordingly, it becomes possible to execute a sequential operation through high-speed processing such as pipeline, super-scaler, or out-of-order.

Further, it is preferable that the control program

is a ladder diagram or an instruction list generated from
the ladder diagram, and the control-program dividing unit
divides the control program into a plurality of blocks at
a predetermined rung in the ladder diagram to generate a
5   program file every block concerned.   Accordingly, it
becomes possible to manage program files every a series of
circuit blocks from input up to output and lower the frequency
of a trouble that optimum assignment of register variables
having input/output device values under compiling is impeded
10   due to division. Particularly, it is possible to improve
the register utilization efficiency by a compiler and
efficiently use an optimizing function of minimum operation
assignment by constituting a block of a collection of a
plurality of rungs.

15   Further, it is preferable that the control program
is a ladder diagram or an instruction list generated from
the ladder diagram, and the control-program dividing unit
divides the control program into a plurality of blocks at
a predetermined rung serving as a jump destination for a
20   jump instruction in the ladder diagram to generate a program
file every block concerned. Accordingly, optimization of
a compiler is not influenced and it is possible to manage
a divided program file as a subroutine.

       Further, it is preferable that the control program
25   is a ladder diagram or an instruction list generated from

17

the ladder diagram, the control-program dividing unit extracts all or some of rungs including instructions for a common input or output device from the ladder diagram, one block is constituted of all or some of the extracted

5   rungs, and a program file is generated every block concerned. Accordingly, when it is necessary to modify a control program for a specific device, it is possible to obtain an execution code by changing and recompiling only a program file corresponding to the portion to be modified.

10   Further, it is preferable that an optimization filtering unit is included which reconstructs the control program into an optimum code system by rearranging codes for excluding variables not cited and redundant codes and locally arranging instructions for a common input or output

15   device and a control program optimized by the optimization filtering unit is newly used as the former control program. Accordingly, it becomes possible to provide a control program constituted of optimized codes for the compiler.

Further, it is preferable that a relating table is

20   included which relates a sample program whose processing time is already known with a control program corresponding to the execution code, wherein processing time roughly-estimating means for estimating the sequential-processing execution time of a programmable

25   controller is included. Accordingly, it becomes possible

to know the processing time of each instruction in a control program, which has been difficult so far.

The control-program-development supporting apparatus according to still another aspect of the present invention comprises a control-program converting unit which converts the control program into an advanced control program described with a universal-computer-readable advanced language, a debugging-code generating unit which generates a debugging control program by inserting a line number concerned into a portion corresponding to each line constituting the instruction list in a source code constituting the advanced-language control program, and a debugging executing unit which relates each line of the instruction list with the execution part of the advanced-language control program and displaying them by step-executing the debugging control program.

According to the above invention, it is possible to step-execute an instruction list and the execution part of an advanced-language control program obtained by converting the instruction list display by relating the instruction list with the execution part and display the execution part. Therefore, also in the case of a programmable controller to be operated in accordance with an execution code obtained by compiling an instruction list, it is possible to confirm operations at the step-execution level of an instruction

list described by a user.

The control-program-development supporting apparatus according to still another aspect of the present invention comprises a first storing unit which stores the execution code, a second storing unit which stores the data for the difference between the execution code which is stored in the first storing unit and a new execution code, a microprocessor to be directly executed in accordance with the execution code, and a patch processing unit which changes an execution code currently executed to a new execution code and continuously executing the changed execution code at a predetermined timing.

According to the above invention, in accordance with the data for the difference between the execution code which is continuously executed and the new execution code, an execution code currently executed is changed to a new execution code at a predetermined timing and the changed execution code is continuously executed. Therefore, even if it is necessary to change execution codes, it is possible to correct only a changed binary code among execution codes and change it to a new execution code and continue a new sequential operation without interrupting the execution state of a programmable controller.

The control-program-development supporting apparatus according to still another aspect of the present

invention comprises a storing unit which stores the execution code and a microprocessor to be directly executed by the execution code. The execution code includes binary data generated by compressing the control program.

5   According to the above invention, a control program necessary for development or debugging and execution codes are unified and stored in a programmable controller. Therefore, when loosing the control program or directly operating the control program from a job-site programmable

10   controller, it is possible to fetch the control program from the execution codes.

The   control-program-development   supporting apparatus according to still another aspect of the present invention comprises a compressing unit which compresses the

15   control program to generate a compressed file, a code converting unit which generates compressed data obtained by converting the compressed file into the code system of the control program, and a compiler for combining the control program with the compressed data and compiling the program

20   and the data into codes directly executable by a programmable controller.

According to the above invention, a control program necessary for development or debugging and execution codes to be actually executed are unified to generate a new

25   execution code.   Therefore, when loosing the control

21

program or directly operating the control program from a job-site programmable controller, it is possible to fetch the control program from the execution codes.

The programmable controller according to still another aspect of the present invention comprises a storing unit which stores the control program, an instruction counting unit which counts the appearance frequency of each instruction used for the control program, a pattern-matching-table generating unit which generates a pattern matching table in which instructions are listed starting with an instruction having the highest appearance frequency in accordance with results counted by the instruction counting unit, and an interpreting unit which executes the control program while pattern-matching the control program in order of the instructions listed in the pattern matching table and interpreting the control program into execution codes directly executable by the programmable controller.

According to the above invention, instructions in a control program are pattern-matched starting with an instruction having the highest appearance frequency when interpreting the control program into execution codes. Therefore, it is possible to quickly find an execution code suitable for each instruction of the control program when interpreting the control program.

The control-program-development supporting apparatus according to still another aspect of the present invention comprises an instruction counting unit which counts the appearance frequency of each instruction used for the control program, a pattern-matching-table generating unit which generates a pattern matching table in which instructions are listed starting with an instruction having the highest appearance frequency in accordance with results counted by the instruction counting unit, and a compiler which compiles the control program into codes directly executable by the programmable controller while pattern-matching the control program in order of instructions listed in the pattern-matching table.

According to the above invention, because instructions are pattern-matched starting with an instruction having the highest appearance frequency in a control program when compiling the control program into execution codes, it is possible to quickly find an execution code suitable for each instruction of the control program when compiling the control program.

The control-program-development supporting apparatus according to still another aspect of the present invention comprises any of the above-mentioned control-program-development supporting apparatus according to the present invention.

According to the above invention, it is possible to realize functions of the above control-program-development supporting apparatus and those of the above programmable controller with a software programmable controller making it possible to develop a control program and also execute sequential processing.

Other objects and features of this invention will become apparent from the following description with reference to the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a block diagram showing a schematic configuration of a programmable controller of first embodiment;

Fig. 2 is an illustration for explaining the generation of an execution code in a control-program-development supporting apparatus of the first embodiment;

Fig. 3 is an illustration for explaining the generation of an execution code followed by division of an control program in a control-program-development supporting apparatus of second embodiment;

Fig. 4 is an illustration showing a ladder diagram;

Fig. 5 is a table showing the relation between the number of steps and a divided file in the control-program-development supporting apparatus of second

embodiment;

Fig. 6 is a flowchart showing the processing by the rung dividing method in the control-program-development supporting apparatus of the second embodiment;

5   Fig. 7 is a table showing the relation between a jump-destination label name and a divided file in the control-program-development supporting apparatus of the second embodiment;

Fig. 8 is a flowchart showing the processing by the
10   jump-destination dividing method in the control-program-development supporting apparatus of the second embodiment;

Fig. 9 is a flowchart showing the processing by the unit dividing method in the control-program-development
15   supporting apparatus of the second embodiment;

Fig. 10 is an illustration for explaining the generation of an execution code when a control program must be modified in the control-program-development supporting apparatus of the second embodiment;

20   Fig. 11 is an illustration for explaining the generation of an execution code followed by division of a control program and conversion of the program into an advanced programming language in a control-program-development supporting apparatus of third
25   embodiment;

25

Fig. 12 is an illustration for explaining the generation of an execution code when a control program to be once converted into an advanced programming language must be modified in the control-program-development supporting apparatus of the third embodiment;

Fig. 13 is an illustration for explaining the generation of an execution code when a converter divides a control program and converts the program into an advanced programming language in the control-program-development supporting apparatus of the third embodiment;

Fig. 14 is an illustration for explaining the generation of an execution code when a control program must be modified when a converter divides the control program and converts the program into an advanced programming language in the control-program-development supporting apparatus of the third embodiment;

Fig. 15 is an illustration for explaining the generation of an execution code of a control-program-development supporting apparatus of fourth embodiment;

Fig. 16 is a table showing sample programs and their sizes, and processing times in a control-program-development supporting apparatus of fifth embodiment;

Fig. 17 is an illustration for explaining the

26

development environment and execution environment of a control program in a control-program-development supporting apparatus of sixth embodiment;

Fig. 18 is an illustration for explaining debugging of the control-program-development supporting apparatus of the sixth embodiment;

Fig. 19 is an illustration for explaining a sampling tracing function in the control-program-development supporting apparatus of the sixth embodiment;

Fig. 20 is an illustration for explaining a status latching function in the control-program-development supporting apparatus of the sixth embodiment;

Fig. 21 is an illustration for explaining a debugging method in the control-program-development supporting apparatus of the sixth embodiment;

Fig. 22A and Fig. 22B are illustrations for explaining the replacement of an execution code using a binary patch in a programmable controller of seventh embodiment;

Fig. 23 is an illustration for explaining the generation of an execution code in a control-program-development supporting apparatus of eight embodiment;

Fig. 24 is an illustration showing a source file that is the content of a control program in the control-program-development supporting apparatus of the

eight embodiment;

Fig. 25 is an illustration showing a C-language expression obtained by converting a compressed file in the control-program-development supporting apparatus of the

5 eight embodiment;

Fig. 26 is an illustration for explaining compiling in a control-program-development supporting apparatus of ninth embodiment; and

Fig. 27 is a flowchart showing the compiling operation

10 of the control-program-development supporting apparatus of the ninth embodiment.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Preferred embodiments of a programmable controller

15 and a control-program-development supporting apparatus of the present invention are described below in detail by referring to the accompanying drawings. However, the present invention is not restricted to the embodiments.

First,   the   programmable   controller   and

20 control-program-development supporting apparatus of first embodiment are described. The programmable controller of the first embodiment mounts a universal microprocessor constructed in accordance with the acceleration mounting art.   The   control-program-development   supporting

25 apparatus of the first embodiment compiles a control program

described with a sequential-processing language such as an instruction list and generates execution codes directly executable by the universal microprocessor.

Fig. 1 is a block diagram showing a schematic configuration of the programmable controller of the first embodiment. The programmable controller 10 comprises a universal microprocessor 11 for executing sequential processing in accordance with execution codes, a memory 14 for storing execution codes and the like, a communication interface 16 for receiving a binary file and the like serving as execution codes from a control-program-development supporting apparatus 20 and transmitting a program and the like in the memory 14 to an external unit according to necessity, depend on the indication of the universal microprocessor 11, and an input/output control section 15 for inputting a signal from an external input device 21 and outputting a control signal to an external output device.

Particularly, as shown in Fig. 1, the universal microprocessor 11 is provided with a CPU core 12 for performing basic arithmetic processing of the stored-program type or the like, and an acceleration mounting art section 13. And the acceleration mounting art section 13 includes an instruction cache and that is constructed. The acceleration mounting art section 13 includes a data cache, a secondary cache, a pipeline logic, and a scaler

29

logic. That is, the universal microprocessor 11 is a microprocessor capable of performing not only pipeline processing and super-scaler processing but also acceleration processing such as out-of-order, branch estimation, and speculative execution. In the case of the universal microprocessor 11 shown in Fig. 1, the acceleration mounting art section 13 is separated from the CPU core 12. However, a configuration is also present which has a plurality of functional blocks of the acceleration mounting art section 13 in the CPU core 12.

Moreover, the memory 14 is used for arithmetic processing by the universal microprocessor 11 and constituted of a work memory for loading actual execution parts or the whole of execution codes, a ROM in which a BIOS (Basic Input Output System) is recorded, and a nonvolatile memory such as a flash memory for storing every execution code. Though not illustrated, the programmable controller 10 can be provided with a secondary storing unit such as a magnetic disk drive or memory card for storing execution codes, programs, or data.

That is, the configuration of a conventional programmable controller can be applied to the programmable controller 10 shown in Fig. 1 except that a microprocessor constituted of a conventional ASIC is replaced with a universal microprocessor provided with acceleration

mounting means.

Thus, to make the programmable controller 10 execute sequential processing, execution codes directly executable by the universal microprocessor 11 are necessary, which are generated by the control-program-development supporting apparatus 20.

The control-program-development supporting apparatus 20 can be realized by the same configuration as a universal computer system such as a personal computer as long as the control-program-development supporting apparatus 20 is provided with an interface communicable with the programmable controller 10.

Particularly, the control-program-development supporting apparatus 20 is provided with a compiler for reading the file of a control program described with a ladder diagram or instruction list and compiling the code system of the read file into execution codes directly executable by the universal microprocessor 11. Fig. 2 is an illustration for explaining the generation of an execution code by the control-program-development supporting apparatus of the first embodiment.

As shown in Fig. 2, the control-program-development supporting apparatus 20 generates an execution code 9 by inputting a control program 1 described with a sequential-processing language such as a ladder diagram or

instruction list already developed or newly developed to a compiler 100. Though the compiler 100 can be realized by a software program or hardware logic, it is necessary to generate an execution code for maximally exhibiting functions of the universal microprocessor 11 by using an art for optimizing the above universal microprocessor.

As described above, it is possible to constitute the programmable controller of the first embodiment by using the universal microprocessor 11 comparatively inexpensive and easily available without designing or developing an exclusive microprocessor as an ASIC and execute sequential processing at a higher speed in accordance with the acceleration mounting art of the universal microprocessor 11.

Moreover, the control-program-development supporting apparatus of the first embodiment makes it possible to obtain the execution code 9 directly executable by the above universal microprocessor 11 by compiling a control program described with a sequential-processing language. Therefore, it is possible to make a programmable controller which mounts the universal microprocessor 11 execute sequential processing by transferring the execution code 9 to the controller and effectively use resources of past control programs. Moreover, it is possible to obtain the execution code 9 by replacing a compiler 100 to be used

between programmable controllers that mount different types of universal microprocessors 11. Therefore, it is unnecessary to develop a control program for each maker or model of a programmable controller, uniformly manage control programs describing the same-content sequential processing, and reduce the development time and debugging time.

The control-program-development supporting apparatus of second embodiment is described below. The control-program-development supporting apparatus of the second embodiment makes it possible to divide a control program described with a sequential-processing language such as an instruction list into a plurality of program files at a predetermined position and corrects and compiles only each of the divided program files.

Fig. 3 is an illustration for explaining the generation of an execution code followed by division of a control program by the control-program-development supporting apparatus of the second embodiment. As shown in Fig. 3, the control-program-development supporting apparatus divides a control program 1 described with a sequential-processing language such as a ladder diagram or instruction list already developed or newly develop and conventionally handled as one program file into a plurality of division programs 3-1 to 3-N in accordance with a division algorithm to be mentioned later.

33

Then, the division algorithm is described below. A control program is divided in order to shorten the time necessary for recompiling when a control program should be modified. However, division should not be performed which greatly impedes the optimization to be performed by the compiler 100. When dividing a control program, it is impossible to obtain a completely optimum execution code compared to a case of compiling a control program not divided at all. Therefore, it is necessary to output a code close to a completely optimum execution code, that is, a code for minimizing the degradation of an optimization level when dividing a control program.

Therefore, for the division algorithm, the following three methods are considered. That is, (1) a method of performing division at a gap between rungs (hereafter referred to as rung dividing method), (2) a method of performing division at a subroutine serving as a jump destination (hereafter referred to as jump-destination dividing method), and (3) a method of performing division of putting input devices or output devices used together every unit (hereafter referred to as unit dividing method).

Particularly, because these three types of division algorithms are based on the expression of a ladder diagram, the ladder diagram is first described below. Fig. 4 is an illustration showing a ladder diagram. In Fig. 4,

34

instructions CM1 to CM6 correspond to commands used for sequential operations. Particularly, in Fig. 4, instructions CM1, CM2, and CM5 denote inputs, CM3 and CM6 denote outputs, and CM4 denotes arithmetic processing.

5 Moreover, in the case of a ladder diagram, a collection of instructions from input to output is expressed as one circuit block that is referred to as a rung.

Fig. 4 shows two rungs R1 and R2. Moreover, the number of steps showing the processing time of an instruction by
10 a specific unit is provided for each rung. In the case of the rung R1 in Fig. 4, the number of steps at the head of the next rung R2 is set to 5 because one step is necessary for read of data to an address assigned into an input port, one step is necessary for write of data in an address assigned
15 to an output port, and two steps are necessary for a MOV instruction for moving data between addresses.

That is, in the case of the rung R1, memories X1 and X2 are used as input devices and a memory Y1 is used as an output device. Moreover, in the case of the rung R2, a memory
20 X1 is used as an input device and a memory Y10 is used as an output device.

(1) rung dividing method

The rung dividing method is a method for dividing a control program at a gap between rungs expressed by the above
25 ladder diagram. By dividing the control program at a gap

between rungs, it is possible to effectively use the arrangement of devices to a register and control deterioration of the optimization effect due to division.

It is possible to consider sequential processing as a collection of rungs and it is the rung processing to perform a sequential operation given to the information of an input device and obtain the value of an output device. Therefore, to accelerate sequential processing, it is necessary to accelerate the processing in each rung. Moreover, when dividing one rung at its middle, it is impossible to effectively use operation results up to the middle of rungs set in a register. Therefore, it is preferable to divide a control program at a gap between rungs.

Therefore, a high-speed operation is realized by arranging input and output devices in a register as many as possible by using the compiler 100. That is, it is necessary to positively effectively use a register by using the optimization function of a compiler.

However, when minutely dividing a control program every gap between rungs, the degree of optimization is lowered because the optimization function for outputting an out-of-ordered code is not used by the compiler 100 due to the optimum-instruction scheduling which is one of acceleration mounting arts in order to generate execution codes of the universal microprocessor described for the first

36

embodiment.

Thus, it is preferable that the divided control programs, that is, the division programs 3-1 to 3-N respectively have a certain-degree size. Therefore, a position for division is designated every specified number of steps. For example, a control program is divided at a gap between rungs every multiple of 100 steps. When the position of a step of a multiple of 100 steps is not a gap between rungs, a control program is divided at a gap between rungs immediately after the position.

The division programs 3-1 to 3-N thus generated are respectively managed as an individual program file so that these division programs can be independently compiled. Fig. 5 is a table showing the relation between the number of steps and divided files. As shown in Fig. 5, each division program is provided with a divided-file name and moreover, the number of steps corresponding to a rung of the original control program 1 is assigned to each file.

When the control program must be corrected, the control program 1 before divided is corrected on, for example, a ladder diagram. The above table is effectively used to extract the number of steps at an actually-corrected part of the control program 1 and specify the file name of a division program including the corrected part.

Then, a specific processing flow of the rung dividing

37

method is described below. Fig. 6 is a flowchart showing
the processing of the rung dividing method. In Fig. 6,
dividing means such as a computer program for executing the
rung dividing method reads one line of converting an
instruction list (hereafter referred to as IL) serving as
the control program 1, particularly, an IL file that is
obtained by conversion of a ladder diagram (step S101). When
the read one line shows the end of the IL file, the processing
is completed (Affirmative in step S102).

However, when the read one line does not show the end
of the IL file (Negative in step S102) but the line corresponds
to a designated number of steps or more and it is present
at a gap between rungs (Affirmative in step S103), a new
file is generated and set to a current file (step S104).
Whether one line is not present at a gap between rungs or
not is determined in accordance with the type of an
instruction. That is, a position where an instruction to
be input appears immediately after an instruction to be
output serves as a gap between rungs. Moreover, a file to
be newly generated is provided with a unique name.

Then, the read one line is written in an already-set
current file (step S105) to repeat the processings in and
after step S101 again. That is, when the read one line is
not present at a gap between rungs (Negative in step S103)
or when the one line does reach a designated number of steps,

the one line is successively written in the current file. Thereby, a file is generated at a gap between rungs every range of a predetermined number of steps. This file is the divided file generated correspondingly every above division program.

(2) Jump-destination dividing method

The jump-destination dividing method is a method for dividing a control program at a gap between rungs expressed by the above ladder diagram and also at a subroutine serving as the jump destination of a jump instruction. Thereby, it is possible to consider the optimization that influences the jump instruction.

Fig. 7 is a table showing the relation between a jump-destination label name and a divided file. As shown in Fig. 7, each division program is provided with a divided-file name and moreover, a label name showing the jump destination of a jump instruction is assigned to each file.

The table shown in Fig. 7, similar to the table shown in Fig. 5, is also effectively used to extract the label name of a subroutine including an actually corrected part in the control program 1 and specify the file name of a division program corresponding to the subroutine.

Then, a specific processing flow of the jump-destination dividing method is described below. Fig.

39

8 is a flowchart showing the processing of the jump-destination dividing method. In Fig. 8, dividing means of a computer program or the like for executing the jump-destination dividing method first reads one line of an IL serving as the control program 1, particularly an IL file obtained by converting a ladder diagram (step S201). When the read one line shows the end of the IL file, the processing is completed (Affirmative in step S202).

However, when the read one line does not show the end of the IL (Negative in step S202) but it shows a gap between rungs and a jump-destination label (Affirmative in step S203), a new file is generated and set to a current file (step S204). A method for determining whether the line shows a gap between rungs and a method for deciding a file name are the same as the case of the above rung dividing method.

Then, the read one line is written in an already-set current file (step S205) and processings are repeated again starting with step S201. That is, when the read one line does not show the gap between rungs or the head of a jump-destination subroutine, the one line is successively written in the current file. Thereby, a file is generated every while the label name of a jump destination is attached. This file is a divided file generated correspondingly every division program described above.

(3) Unit dividing method

The unit dividing method is described below. In the case of sequential processing, an input device and an output device used are generally different every a plurality of units constituting a fabrication line. Change of control programs performed at a fabrication site has a characteristic that the change greatly depends on a unit such as change of input devices connected to the unit. Therefore, in the case of the unit dividing method, rungs corresponding every unit are put together and output to a file.

Then, a specific processing flow of the unit dividing method is described below. Fig. 9 is a flowchart showing the processing of the unit dividing method. In Fig. 9, rungs showing a gap between rungs and the sequential processing in the same unit are first extracted and classified. Then, a plurality of rungs classified for each unit are put together to generate a file corresponding to each unit (step S301).

The numbers of input devices and output devices used for these files are calculated (step S302). When the calculated number of devices is a designated number of devices or less, the files generated in step S301 are assumed as a result of dividing the control program 1 into a plurality of division programs 3-1 to 3-N (Affirmative in step S303). However, when a calculated number of devices is larger than the designated number of devices, classification of units is repeated again in step S301 by narrowing down the range

41

of the number of rungs serving as a criterion of division
(negative in step S303).

Because some devices for handling sensor information
may be used for a plurality of units, it is difficult to
5   completely classify the devices by rungs so that the units
are not duplicated. However, as the processing shown by
the above flowchart, by controlling the number of devices
used between divided files after division, it is possible
to control the reduction of the optimization effect described
10   for the rung dividing method. Moreover, it is allowed for
a user to classify rungs every unit described above.

Therefore,   the   control-program-development
supporting apparatus of the second embodiment can use any
one of three types of division algorithms as described above.
15   The file of a plurality of division programs generated with
the above algorithm is effectively used when the control
program 1 must be corrected.

Fig. 10 is an illustration for explaining the
generation of an execution code when the control program
20   1 must be corrected. As shown in Fig. 10, for example, when
a portion 2 is modified due to correction of the control
program 1, only a file corresponding to the division program
3-1 including the modified portion 2 is recompiled by the
compiler 100 to obtain an object file of only the portion.
25   The object file is replaced with an object file corresponding

42

to the division program 3-1 before corrected.

The newly generated object file is linked with other already-generated object file, that is, an object file corresponding to a not-modified division program by a linker and thereby, a new execution code 9 is generated.

As described above, according to the control-program-development supporting apparatus of the second embodiment, it is possible to obtain an execution code by dividing the control program 1 described with a sequential-processing language such as an instruction list into a plurality of programs at a position not influencing compiling and compiling each division program. Therefore, even if the control program 1 must be corrected, only a division program corresponding to the portion changed due to the correction is recompiled. Therefore, it is possible to shorten the compiling time when the correction occurs. Particularly, at a job site where a programmable controller is set, parts of the control program 1 to be corrected are concentrated on a specific portion. Therefore, when a control-program-development supporting apparatus such as a handy program loader is brought into a job site to perform maintenance, it is possible to shorten the maintenance time.

The above-described compiler 100 can use a compiler for generating an execution code of a universal microprocessor constructed by the acceleration mounting art

43

as described for the first embodiment.

The control-program-development supporting apparatus of third embodiment is described below. The control-program-development supporting apparatus of the third embodiment obtains an execution code by converting a plurality of division programs generated by the second embodiment into files respectively described with an advanced programming language by a converter and compiling a file of the advanced programming language.

Fig. 11 is an illustration for explaining the generation of an execution code followed by division of a control program and conversion to an advanced programming language. As shown in Fig. 11, processings until dividing the control program 1 into a plurality of division programs 3-1 to 3-N are the same as the case of the second embodiment.

The control-program-development supporting apparatus of the third embodiment is different from the control-program-development supporting apparatus of the second embodiment in that the division programs 3-1 to 3-N are input to a converter 101 and converted into files of a plurality of advanced programming languages by the converter 101. Particularly, Fig. 11 shows a case of using C language as an advanced programming language, in which the division programs 3-1 to 3-N are converted into files 4-1 to 4-N respectively described with C language.

The files 4-1 to 4-N described with C language generated by the converter 101 are respectively handled as an independent program file and input to a C-language compiler 102. The C-language compiler 102 respectively compiles these program files described with C language to generate object files and thereby, the execution code 9 is obtained. The execution code 9 is actually generated by linking a plurality of object files generated by the C-language compiler 102 by a not-illustrated linker.

Fig. 12 is an illustration for explaining the generation of an execution code when the control program 1 must be corrected. As shown in Fig. 12, when a part corresponding to a modified portion 2 of the control program 1 must be corrected, it is possible to modify the program file 4-1 described with C language corresponding to the division program 3-1 including the modified portion 2 by directly operating the program file 4-1. When a correction range is large, it is also allowed to generate a division program gain by directly modifying the control program 1.

When the correction is completed, only the modified program file 4-1 described with C language is recompiled by the C-language compiler 102 to obtain an object file corresponding to the division program 3-1. The object file is replaced with an object file corresponding to the division program 3-1 before corrected.

Moreover, the above newly-generated object file is linked with an object file corresponding to other already-generated object file, that is, an object file corresponding to a division program not modified by a linker and thereby, a new execution code 9 is generated.

In the case of examples shown in Figs. 11 and 12, it is assumed that the control program 1 is already divided into a plurality of division programs 3-1 to 3-N at the front stage of the converter 101. However, it is also allowed to divide the control program 1 when it is converted into an advanced programming language such as C language without generating the division programs 3-1 to 3-N.

Fig. 13 is an illustration for explaining the generation of an execution code when a converter divides the control program 1 and converts the program 1 into an advanced programming language. As shown in Fig. 13, no division program is generated but a converter 103 divides the control program 1 into a plurality of program files 4-1 to 4-N respectively described with C language. These program files 4-1 to 4-N have the same contents as those shown in Fig. 11. That is, as the internal processing of the converter 103, the converter 103 once divides a control program into a plurality of division programs in accordance with the code system of an instruction list and converts each division program into an advanced programming language.

Fig. 14 is an illustration for explaining the generation of an execution code when the control program 1 must be corrected in a case in which a converter divides the control program 1 and converts the program 1 into an advanced programming language. As shown in Fig. 14, when a part corresponding to a modified portion 2 of the control program 1 must be corrected, it is possible to modify the program file 4-1 described with C language corresponding to a division program including the modified portion 2 by directly operating the file 4-1. When the correction range is large, it is also allowed to generate a re-divided program file described with C language by directly modifying the control program 1.

When the correction is completed, only the modified program file 4-1 described with C language is recompiled by the C-language compiler 102 to obtain an object file. Generation of a new execution code 9 is described for Fig. 12.

As described above, the control-program-development supporting apparatus of the third embodiment obtains an execution code by dividing the control program descried with a sequential-processing language such as an instruction list into a plurality of programs at a position not influencing compiling and moreover converting the division programs into an advanced programming language such as C language.

47

Therefore, it is possible to enjoy advantages obtained with the second embodiment. Moreover, when a correction is necessary, it is possible to obtain a purposed execution code by modifying a program file described with an advanced programming language and further shorten the time necessary for development or debugging of a control program.

The control-program-development supporting apparatus of fourth embodiment is described below. The control-program-development supporting apparatus of the fourth embodiment obtains a compact and optimum execution code from a compiler by once reconstituting codes with an optimization filter before inputting a control program such as an instruction list to the compiler or converter described in connection to the first to third embodiments.

Fig. 15 is an illustration for explaining the generation of an execution code of the control-program-development supporting apparatus of the fourth embodiment. As shown in Fig. 15, optimization processing is applied to a control program 1 by an optimization filter 104 before inputting the program 1 to a compiler 105. The optimization filter 104 performs optimization without modifying the sequential-processing content of the original control program 1 and outputs an optimized control program 5 by excluding not-cited variables and redundant codes, recombining logics, and replacing

48

sequences for locally arranging instructions for common input and output devices. Particularly, for optimization, by collecting devices to be accessed and instructions to be used nearby in the control program 1, it is possible to provide locality for a program and improve the execution efficiency of an execution code after compiled.

Because the execution-code generation shown in first to third embodiments can be applied to the flow of the processings until the execution code 9 is generated from the optimized control program 5, the description of this processing is omitted.

As described above, the control-program-development supporting apparatus of the fourth embodiment reconstitutes the code system of the control program 1 described with a sequential-processing language such as an instruction list before inputting the program 1 to the compiler 105, it is possible to generate more-compact and optimum execution code 9 by the rear-stage compiler 105.

The control-program-development supporting apparatus of fifth embodiment is described below. The control-program-development supporting apparatus of the fifth embodiment prepares a sample program whose execution time is already known and estimates an actual execution time of an instruction or the like in development of a control program before compiled.

Fig. 16 is a relating table showing instructions, in the control-program-development supporting apparatus of fifth embodiment, that is, sample programs, their sizes, and their processing times. As shown in Fig. 16, various instructions are used and samples of control programs of various sizes and their execution times are prepared. In general, it is difficult to estimate the processing time of an execution code generated by being compiled. Therefore, a program having a processing time closest to the above processing time is selected out of prepared sample programs and used as a criterion of the processing time of a generated control program. Estimation of processing time using the relating table shown in Fig. 16 can be also automatically performed with a computer program by the fact that one of the instructions shown in Fig. 16 determines a portion including by a predetermined rate as a portion approximated to the instruction.

As described above, according to the control-program-development supporting apparatus of the fifth embodiment, it is possible to estimate an actual execution time when developing or debugging the control program 1 such as an instruction list or the like. Therefore, it is easy to improve the processing speed of the whole sequential processing and obtain a time table.

The control-program-development supporting

apparatus of sixth embodiment is described below. The control-program-development supporting apparatus of the sixth embodiment makes it possible to know a debugging portion of an advanced programming language corresponding to a portion of an original control program while the language is debugged in a processing system for once converting a control program described with a sequential-processing language such as an instruction list into an advanced programming language such as C language as described for the above third embodiment.

In this case, a control-program development environment for obtaining an execution code is first described. Fig. 17 is an illustration for explaining a development environment and an execution environment of a control program. As shown in Fig. 17, a development environment 30 is constituted of an editor 106, a converter 101, a debugger 110, a compiler 108, and a linker 107. The editor 106 serves as a tool for generating the control program 1 of a ladder diagram, an instruction list, or the source list of an advanced programming language obtained by converting the ladder diagram and the instruction list.

Because the converter 101, compiler 108, and linker 107 are the same tools as those described in connection to the first to fifth embodiments, their descriptions are omitted. The debugger 110 is a tool for debugging a generated

control program. An execution environment 40 is a section for actually executing the execution code 9 generated in the development environment 30, which is allowed to use a compiling-type programmable controller or an emulator obtained by simulating the programmable controller on a control-program-development supporting apparatus.

Because the control-program-development supporting apparatus of the sixth embodiment particularly has functions and advantages for debugging, the debugger 110 is described below in detail. Fig. 18 is an illustration for explaining the debugging operation. The debugger 110 has a monitoring function for each device described in the control program 1 currently executed, a step-executing function for step-executing a control program, a sampling tracing function, a status latching function, a breaking function for breaking when a variable (that is, device) shows a designated value, and a breakpoint setting function for setting a breakpoint in a control program.

The memory 41 shown in Fig. 18 is equivalent to the memory 14 of the programmable controller described in connection to the first embodiment. In general, in the case of a control-program-development supporting apparatus, the execution environment 40 is frequently realized by the above emulator. In this case, the memory 41 serves as a memory mounted on a control-program-development supporting

apparatus.

The monitoring function is a function for referring to the value of a device while the control program 1 is executed and displaying it with graphical means such as a numerical value or graph. Moreover, the monitoring function makes it possible to rewrite the value of a device under execution.

Furthermore, the sampling tracing function is a function for recording the value of a device designated every scan. Fig. 19 is an illustration for explaining the sampling tracing function. As shown in Fig. 19, the sampling tracing function makes it possible to generate the timing chart of a designated device (in Fig. 19, output devices Y0 and Y1 are shown). In the sampling tracing, it is possible to secure a region for recording the value of a device in the debugger 110 and use a memory not currently used in an execution environment. Moreover, it is allowed to secure a sampling-tracing memory in an execution environment.

The status latching function is a function for recording values of all devices at a certain moment under execution. Fig. 20 is an illustration for explaining the status latching function. The timing (that is, trigger) for a device-value recording function to record the value is the time when a designated device reaches a designated value or it is possible for a user to manually input the timing from the debugger 110. When a trigger works, the

content of a memory is copied to a status latching memory 42. The memory content stored in the status latching memory 42 can be monitored by the debugger 110.

The debugging method that is a feature of this embodiment is described below. Fig. 21 is an illustration for explaining the debugging method. Hereafter, a system is considered which generates the execution code 9 by converting an input source file described with an instruction list, that is, the control program 1 into an advanced-language converting file 4 described with an advanced programming language such as C language by the converter 101 and compiling the advanced-language converting file 4 by the compiler 108.

First, the converter 101 inserts the information for line number of each instruction list constituting the control program 1 into a corresponding portion in the advanced-language converting file 4 when converting the control program 1 into the advanced-language converting file 4. As illustrated, this processing can be realized by a compiler instruction "#line" for recognizing line numbers of the source file of a general C-language processing system.

The compiler instruction "#line" is a special instruction for a compiler, that each line of the source file of C language corresponds to each line of another source file. General C-language compilers such as Gnu C Compiler

of Free Software Foundation and Visual C/C++ Compiler of Microsoft Corp. also support the above function and the syntax of them is shown as #line line number "source file name".

5   Thus, by embedding a compiler instruction corresponding to each line of an original instruction-list source and compiling the instruction by the compiler 108 such as the above Gnu C or Visual C, it is possible to obtain an executable-type object file in which the line-number information of an input source file is embedded, that is,
10   the execution code 9.

By step-executing the execution code 9 on a Gnu Debugger (gdb) in the case of Gnu C or a debugger attached to the integrated development environment VisualStudio of
15   Microsoft Corp. in the case of Visual C, the original control program 1 is read on each debugger and it is possible to step-execute the control program 1 every line on the program 1.

As described above, according to the
20   control-program-development supporting apparatus of the sixth embodiment, the description of a part corresponding to each instruction of the original control program 1 is inserted into the advanced-language converting file 4 of the advanced programming language obtained by converting
25   the control program 1 described with a sequential-processing

55

language such as an instruction list. Therefore, the control-program-development supporting apparatus of the sixth embodiment is effective particularly when a developer of a control program fully knows an advanced programming language such as C language though he/she is inexperienced in the description of an instruction list and vice versa and it is possible to shorten the developing time and debugging time of the control program.

Moreover, when correcting a program file described with an advanced programming language such as C language, it is possible to modify an original control program such as an instruction list at the same time. Therefore, it is possible to improve the consistency between the program file and the control program.

The programmable controller of seventh embodiment is described below. The programmable controller of the seventh embodiment receives and stores a binary patch generated in accordance with the difference data between control programs before and after modified to execute the binary patch and thereby, replaces an execution code currently execute with a new execution code at a predetermined timing of the sequential processing currently executed and continues the sequential processing according to the new execution code.

Fig. 22A and Fig. 22B are illustrations for explaining

execution-code replacement using a binary patch in the programmable controller of the seventh embodiment. First, the processing for the programmable controller to receive and execute a new execution code is described below.

As shown in Fig. 22A an execution code 8 is first generated by the control-program-development supporting apparatus 20. Because generation of an execution code by the control-program-development supporting apparatus 20 is the same as the case of a conventional control-program-development supporting apparatus or the control-program-development supporting apparatuss of the first to sixth embodiments, its description is omitted.

The programmable controller 10 receives the execution code 8 through a not-illustrated communication interface and reads the code 8 in a memory 14 such as a work memory by a program loader 120. Thereby, the program controller 10 can start the sequential processing according to the execution code 8.

It is assumed that a control program 1 serving as the original source file of the execution code 8 is modified and resultantly a control program 1' is generated. Particularly, it is assumed that this modification is a local modification of the control program 1 before modified but completely new contents are not written from the beginning. In this case, an execution code 8' corresponding

to the control program 1′ is generated similarly to the case of the above execution-code generation. Then, the control-program-development supporting apparatus 20 reads the held execution code 8 before modified and generates the difference data between the execution code 8 and the modified execution code 8′ with a difference-data generating tool 130.

The execution codes 8 and 8′ are not text data shown by ASCII codes but they are object files shown by binary codes. Therefore, the difference between these execution codes as binary data is local such as the difference between the original control programs 1 and 1′. For example, if the compiler 100 only converts each line of the control program 1 before modified into a corresponding block position in an object file, the difference has only a value when a block is newly added to the nth block position or the nth block position is newly replaced with the above block.

Moreover, the difference is only a difference between test segments, data segments, or bss segments (not-initialized segments) in each execution code. The difference-data generating tool 130 extracts the above difference and outputs it as a file and the output difference data is referred to as a binary patch 7.

A patch denotes a technique for reproducing original data from a difference between files and its difference data

generally used in the computer field, which generally generates a new text file in accordance with a difference between the so-called text files of the ASCII code. The binary patch performs the above operations for binary data.

However, the binary patch is applied to a file set on the so-called secondary memory such as a hard disk of a computer but it is not means to be applied to a program in a memory currently executed. However, the programmable controller of the seventh embodiment applies the binary patch 7 to an execution code under execution.

Therefore, the programmable controller 10 receives not the newly-generated execution code 8 but the binary patch 7 smaller than the code 8 in size through a not-illustrated communication interface and stores the code 8 in a nonvolatile memory such as a flash memory.

In this case, the program loader 120 on the programmable controller 10 also functions as a patcher for applying a patch, reads information from the binary patch data 7 at a proper timing in repetitive looping of the execution code 8 under execution, and replaces codes of the execution code 8 under execution. That is, as described above, the program loader 120 performs insertion of a block into a certain block position, rewriting of a block, or insertion and rewriting for each segment. Moreover, when rewriting of an execution code is completed, the programmable controller 10

immediately executes the new execution code to continue sequential processing.

As described above, the programmable controller of the seventh embodiment automatically replaces an execution code under execution in accordance with difference data in which only a modifying portion of a control program is described and continues sequential processing according to a replaced new execution code. Therefore, it is enough to hold the binary patch 7 smaller than a changed execution code in size without holding the whole changed execution code in the programmable controller 10 like the conventional case.

The concept corresponding to modification of a control program by the above difference data can be also applied to a ladder diagram or instruction list in an interpreter-type programmable controller.

The control-program-development supporting apparatus of eight embodiment is described below. The control-program-development supporting apparatus of the eight embodiment inserts a compressed file of a control program before compiled into an execution code obtained by compiling it to unify an execution code and a control program.

Fig. 23 is an illustration for explaining the generation of an execution code by the control-program-development supporting apparatus of the

eight embodiment. First, the control-program-development supporting apparatus converts a control program 1 such as an instruction list into an advanced programming language such as C language with a converter 101a and inputs a program

5   file 4a described with C language obtained through the above conversion to a C-language compiler 102. Then, the C-language compiler 102 compiles the program file 4a described with C language and outputs an object file 8a.

Moreover, the control-program-development

10 supporting apparatus compresses the control program 1 with a file compressing tool 140 to generate a compressed file 99. It is possible to use gzip, bzip2, or lha as the file compressing tool 140. The compressed file 99 compressed by the above tool results in not the text format of ASCII

15 code but binary data.

The control-program-development supporting apparatus converts the compressed file 99 into a program file 4b described with C language with a converter 101b. The compressed file 99 is converted into a C-language

20 expression with the converter 101b in order to include the source code of the compressed control program in a part of an execution code to be finally generated not as a mere attached file.

Fig. 24 is an illustration showing a source file that

25 is the content of the control program 1. Moreover, Fig.

25 is an illustration showing a C-language expression obtained by converting the compressed file 99. The control program 1 expressed with an instruction list shown in Fig. 24 is converted into the compressed file 99 as binary data by the file compressing tool 140 as described above.

When the compressed file 99 is arranged like "01h, 02h, 03h,..." starting with the head data, it is converted into {0x01, 0x02, 0x03,...} which is the arrangement every one-bite data by the converter 101b. The arranged data shown in Fig. 25 is a conversion result to the source file shown in Fig. 24.

Thus, a compressed file is converted into a format which can be interpreted by the C-language compiler 102 to generate the program file 4b expressed by C language. Then, the C-language compiler 102 compiles the C-language-expressed program file 4b to output an object file 8b.

Two object files 8a and 8b generated by the C-language compiler 102 are linked each other by a liner 150 and one directly-executable execution code 90 can be finally obtained by a programmable controller. That is, the execution code 90 is constituted of a part for actually contributing to sequential processing and the information for the source code of the control program 1.

The execution code 90 thus obtained is generally stored

62

in a nonvolatile memory or the like of a programmable controller. As shown in Fig. 25, however, because it is included in an execution code by assuming that the compressed data obtained by compressing the control program 1 has an arrangement with a name of _InputSource and the file name of the data is sample.il, gz, it is possible to extract the compressed file 99 in accordance with the name. Moreover, after the compressed file 99 is extracted, it is possible to restore the original control program 1 by using a developing tool corresponding to a compression format such as gunzip or bunzip generally provided for the file compressing tool 140.

As described above, the control-program-development supporting apparatus of the eight embodiment generates an execution code in which a part to contribute to actual sequential processing and the source-code information of the control program 1 such as an instruction list are united into one body. Therefore, it is possible to unify a control program and an execution code and store them in a programmable controller. Thus, when loosing a control program necessary for development or debugging or directly operating a control program from a programmable controller at a job site, it is possible to fetch the control program from an execution code and realize the consistency between the execution code and the control program.

The programmable controller and control-program-development supporting apparatus of ninth embodiment are described below. The control-program-development supporting apparatus of the first embodiment performs pattern matching of instructions of a control program starting with an instruction having the highest appearance frequency when interpreting the control program into execution codes while the control-program-development supporting apparatus of the ninth embodiment performs pattern matching of instructions of a control program starting with an instruction having the highest appearance frequency when compiling the control program into execution codes.

Fig. 26 is an illustration for explaining the compiling by the control-program-development supporting apparatus of the ninth embodiment. Fig. 26 particularly shows a system for converting an input source file 1 serving as a control program such as an instruction list described by a user and generating an output file 70 serving as an execution code directly executable by a programmable controller.

In general, an instruction list basically frequently uses an LD instruction and an OUT instruction, in which many biases are present in appearance frequencies of instructions because a user frequently uses a certain instruction when performing processings having the same content. Moreover,

because many programs are diverted because of reliabilities based on their past results, biases of instructions used are also present due to the above mentioned.

Therefore, a program converting tool 160 corresponding to a compiler has a pattern-detecting-and-rearranging section 170. The pattern-detecting-and-rearranging section 170 has a pattern-appearance counting section 171 and a pattern-comparing-and-rearranging section 172. The pattern-appearance counting section 171 counts appearance frequencies of instructions when converting them with each program converting tool 160 and stores the frequencies in a pattern-appearance-frequency storing file 60.

The pattern-comparing-and-rearranging section 172 rearranges first to fourth pattern comparisons 181a to 181d of a pattern matching section 180 to a forward sequence of pattern matching starting with an instruction having the highest appearance frequency out of the pattern-appearance-frequency storing file 60. First to fourth action executing sections 182a to 182d corresponding to the pattern comparisons are naturally interlocked. Thereby, because an instruction having a higher appearance frequency is earlier pattern-matched, the time up to completion of processing is shortened. It is allowed that the rearrangement timing of the pattern-comparing-and-rearranging section 172 is the time

65

when going out of the pattern matching section 180 and returning to a line detecting section 161 or the program converting tool 160 is started.

The compiling operation of the control-program-development supporting apparatus is described below. Fig. 27 is a flowchart showing the compiling operation of the control-program-development supporting apparatus of the ninth embodiment.

First, the control-program-development supporting apparatus inputs an input source file 50 to the program converting tool 160. In this case, it is assumed that the input source file 50 is described with the following instruction list.

LD X0

AND X10

OUT Y0

First, the control-program-development supporting apparatus reads a pattern-appearance-frequency storing file 60 to be described later (step S401) and recognizes instructions starting with an instruction having the highest appearance frequency with the pattern-comparing-and-rearranging section 172 (step S402). Thereby, the pattern-comparing-and-rearranging section 172 rearranges the first to fourth pattern comparing sections 181a to 181d in the pattern matching section 180 and the

first to fourth action executing sections 182a to 182d corresponding to the sections 181a to 181d (step S403).

Then, the line detecting section 161 of the program converting tool 160 detects one line of the input source file 50 (step S404) and delivers the detected one line to a semantic interpreting section 162 when the line is not present at the end position of the input source file 50 (negative in step S405).

In the case of the above instruction list, "LD X0" is first delivered to the semantic interpreting section 162. The semantic interpreting section 162 interprets that "LD X0" is an LD instruction and makes the pattern matching section 180 retrieve the execution part of the LD instruction in order to retrieve the execution part of the ID instruction (step S408).

The pattern matching section 180 performs pattern matching in the sequence of the first pattern comparison 181a, second pattern comparison 181b, third pattern comparison 181c, and fourth pattern comparison 181d in order to find various patterns internally mounted, that is, which is the action of the instruction "LD" in the case of the above example (step S409) and, when the fourth pattern comparison 181d has a content relating to the LD instruction, executes the fourth action executing section 182d which is the operation of the LD instruction (step S410). The fourth

67

action executing section 182d outputs an execution code description corresponding to the LD instruction to the output file 70 and the appearance frequency of the instruction is counted by the pattern appearance counting section 171 (step S411).

When the above processing is completed, the fourth action executing section 182d returns to step S404 to repeat the same processing for "AND X10" which is the next line in the above example. However, when the detected one line is present at the end position of the input source file 50 (affirmative in step S405), the section 182d completes writing in the output file 70 (step S406) and stores a result counted by the pattern appearance counting section 171 in the pattern-appearance-frequency storing file 60 as update data (step S407).

As described above, the control-program-development supporting apparatus of the ninth embodiment performs pattern matching of instructions of a control program starting with an instruction having the highest appearance frequency when compiling the control program into execution codes. Therefore, it is possible to quickly find an execution code suitable for each instruction of the control program when compiling the control program and reduce the latency for compiling the control program.

Operations of the above program converting tool 160

68

can be also applied to a converter for converting a control program such as an instruction list into a program file described with an advanced programming language such as C language. That is, in the above description, a content to be written in the output file 70 is not an execution code but an instruction expression of an advanced programming language.

Moreover, operations of the above program converting tool 160 can be applied to an interpreter-type programmable controller. In this case, the input source file 50 can use an instruction list or intermediate codes obtained by converting the instruction list and actions to be executed by the first to fourth action executing sections 182a to 182d result in actual programmable-logic processings.

Functions realized by the control-program-development supporting apparatuss of the first to sixth, eighth and ninth embodiments and functions realized by the programmable controllers of the first, seventh and ninth embodiments can be also mounted on the so-called software programmable controller for realizing development of a control program and also, sequential processing.

As described above, according to the present invention, a universal microprocessor that mounts an acceleration mounting unit is included and a received execution code is

optimized so that it is executed by the acceleration mounting unit of the universal microprocessor. Therefore, an advantage can be obtained that sequential processing can be more quickly executed compared to the case of a conventional programmable controller for which a microprocessor is provided as an ASIC.

According to another aspect of the present invention, a control program described with a sequential-control language is compiled into an execution code executable by a universal microprocessor that mounts an acceleration mounting unit. Therefore, advantages can be obtained that it is possible to execute sequential processing by a programmable controller which mounts a universal microprocessor used for a personal computer and obtain execution codes corresponding to various universal microprocessors by changing a compiler, and thereby correspond to various programmable controllers without modifying a control program, effectively use past resources of control programs, unify the management of control programs, and make the control-program development time efficient.

According to still another aspect of the present invention, it is possible to obtain final execution codes by dividing a control program into a plurality of blocks and compiling every divided block. Therefore, advantages can be obtained that even if a long control program must

be modified, it is possible to generate an execution code by recompiling only a part of the control program which must be modified without recompiling the whole of the control program like a conventional case and shorten the time required to debug or maintain a control program.

According to still another aspect of the present invention, it is possible to obtain final execution codes by dividing a control program into a plurality of blocks, converting each divided block into a universal-computer-readable advanced language, and compiling a program of the universal-computer-readable advanced language corresponding to each block. Therefore, advantages can be obtained that even if a long control program must be modified, it is possible to generate execution codes by recompiling only a part of the program which must be modified without recompiling the whole of the control program like a conventional case, still use a conventional compiler for a universal-computer-readable advanced language, and shorten the time required to debug or maintain a control program.

According to still another aspect of the present invention, an execution code is generated for a programmable controller that mounts a universal microprocessor used for a personal computer. Therefore, an advantage can be obtained that it is possible to execute sequential processing

at a high speed with such high-speed processing as pipeline, super-scaler, or out-of-order compared to the case of a conventional programmable controller for which a microprocessor is provided as an ASIC.

5   According to still another aspect of the present invention, a control-program dividing unit divides a control program into a plurality of blocks at a gap between predetermined rungs in a ladder diagram and generates a program file every block concerned. Therefore, advantages can be obtained that it is possible to manage program files every series of circuit blocks from input up to output and reduce the frequency of a situation that optimization of registers of input/output devices under compiling is interrupted due to division. Particularly, an advantage can be obtained that it is possible to effectively use the optimization function of a compiler for outputting an out-of-order code by constituting one block of the collection of a plurality of rungs for a microprocessor that mounts acceleration mounting unit.

20   According to still another aspect of the present invention, a control-program diving unit divides a control program into a plurality of blocks at a gap between predetermined rungs and a position serving as a jump destination for a jump instruction and generates a program file every block concerned. Therefore, advantages can be

obtained that optimization of a compiler is not influenced and it is possible to manage divided program files as subroutines and simplify debugging and maintenance of a control program.

According to still another aspect of the present invention, a control-program dividing unit generates a program file every block concerned by extracting all or some of rungs including instructions for a common input or output device in a ladder diagram, uniting all or some of the extracted rungs into one block, and dividing the rungs into a plurality of blocks. Therefore, advantages can be obtained that, when a control program must be modified for a specific device, it is possible to obtain an execution code by modifying and recompiling only a program file corresponding to the part to be modified and shorten the time required to debug or maintain the control program.

According to still another aspect of the present invention, a control program is once reconstructed into an optimum code system when compiling the program by excluding variables not cited and excluding and rearranging redundant codes. Therefore, advantages can be obtained that it is possible to provide a control program constituted of optimized codes for the compiler, reduce the time necessary for compiling the program, and improve the speed for executing the program.

73

According to still another aspect of the present invention, it is possible to estimate a sequential-processing execution time in accordance with a relating table relating a sample program whose processing time is already known with a control program corresponding to an execution code. Therefore, an advantage can be obtained that it is possible to accurately relate the actual processing time of an execution code generated by optimizing a compiler with each instruction in a control program corresponding to the processing time though it has been difficult to relate them so far.

According to still another aspect of the present invention, it is possible to perform step execution by relating an instruction list with the execution part of a control program described with an advanced programming language obtained by converting the instruction list and display the execution part. Therefore, advantages can be obtained that it is possible to confirm operations of a programmable controller to be operated in accordance with execution codes obtained by compiling an instruction list at the step-execution level of an instruction list described by a user and simplify the development and debugging of a control program.

According to still another aspect of the present invention, an execution code currently executed is changed

to a new execution code at a predetermined timing in accordance with the difference data between the execution code currently executed and the new execution code and the changed execution code is continuously executed. Therefore,

5   even if execution codes must be changed, advantages can be obtained that it is possible to make a programmable controller change execution codes and continue an execution state by downloading not a new execution code but difference data smaller than the execution code in size and many memory

10 resources of the programmable controller are unnecessary.

According to still another aspect of the present invention, a control program necessary for development or debugging and execution codes are unified and stored in a programmable controller. Therefore, advantages can be

15 obtained that it is possible to fetch the control program from execution codes and realize the consistency between the execution codes and the control program when loosing the control program or directly operating the program from the programmable controller.

20 According to still another aspect of the present invention, a control program necessary for development or debugging and execution codes to be actually executed are unified and newly generated as execution codes. Therefore, advantages can be obtained that it is possible to fetch the

25 control program from the execution codes and realize the

consistency between the execution codes and the control program when loosing the control program or operating the program from a programmable controller at a job site.

According to still another aspect of the present invention, when interpreting a control program into execution codes, instructions of the control program are pattern-matched starting with an instruction having the highest appearance frequency. Therefore, advantages can be obtained that it is possible to quickly find an execution code suitable for each instruction of the control program when interpreting the program and reduce the control-program execution time.

According to still another aspect of the present invention, when compiling a control program into execution codes, instructions are pattern-matched starting with an instruction having the highest appearance frequency. Therefore, advantages can be obtained that it is possible to quickly find an execution code suitable for each instruction of the control program and reduce the latency for compiling the control program.

According to still another aspect of the present invention, advantages can be obtained that it is possible to realize functions of the above control-program-development supporting apparatus and control-program-development supporting apparatus on a

so-called software programmable controller making it possible to develop a control program and also perform sequential processing.

(Note 1)   A programmable controller which performs sequential processing in accordance with execution codes generated by compiling a control program, comprising:

a first storing unit which stores the execution codes;

a second storing unit which stores the data for the difference between an execution code stored in the first storing unit and a new execution code;

a microprocessor to be directly executed by the execution codes;

a patch processing unit which changes an execution code currently executed to a new execution code at a predetermined timing in accordance with the difference data and continuously executing the changed execution code; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

a control-program dividing unit which divides the control program into a plurality of blocks; and

a compiler which compiles all or some of the blocks into execution codes directly executable by a programmable

controller.

(Note 2)   A   programmable   controller   which   performs
sequential processing in accordance with execution codes
generated by compiling a control program, comprising:

5   a first storing unit which stores the execution codes;

           a second storing unit which stores the data for the
difference between an execution code stored in the first
storing unit and a new execution code;

           a microprocessor to be directly executed by the
10   execution codes;

           a patch processing unit which changes an execution
code currently executed to a new execution code at a
predetermined timing in accordance with the difference data
and continuously executing the changed execution code; and

15   a control-program-development supporting apparatus
that   develops   a   control   program   described   with   a
sequential-control language such as a ladder diagram or
instruction   list,   which   control-program-development
supporting apparatus having,

20   a control-program dividing unit which divides the
control program into a plurality of blocks;

           a control-program converting unit which converts all
or some of the blocks into advanced-language control programs
described   with   a   universal-computer-readable   advanced
25   language every blocks concerned; and

a compiler which compiles all or some of universal-computer-readable advanced programming languages corresponding every above block into directly executable codes by a programmable controller.

5 (Note 3) A programmable controller which performs sequential processing in accordance with execution codes generated by compiling a control program, comprising:

a first storing unit which stores the execution codes;

a second storing unit which stores the data for the
10 difference between an execution code stored in the first storing unit and a new execution code;

a microprocessor to be directly executed by the execution codes;

a patch processing unit which changes an execution
15 code currently executed to a new execution code at a predetermined timing in accordance with the difference data and continuously executing the changed execution code; and

a control-program-development supporting apparatus that develops a control program described with a
20 sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

a control-program converting unit which converts the control program into an advanced-programming-language
25 control program described with a

79

universal-computer-readable   advanced   programming
language;

a debugging-code generating unit which generates a
debugging control program by inserting a line number
concerned into a part corresponding to each line constituting
the instruction list in source codes constituting the
advanced-programming-language control program; and

a debugging executing unit which displays each line
of the instruction list and the execution part of the
advanced-programming-language control program by relating
the former with the latter.

(Note 4)   A programmable controller which performs
sequential processing in accordance with execution codes
generated by compiling a control program, comprising:

a first storing unit which stores the execution codes;

a second storing unit which stores the data for the
difference between an execution code stored in the first
storing unit and a new execution code;

a microprocessor to be directly executed by the
execution codes;

a patch processing unit which changes an execution
code currently executed to a new execution code at a
predetermined timing in accordance with the difference data
and continuously executing the changed execution code; and

a control-program-development supporting apparatus

80

that develops a control program described with a
sequential-control language such as a ladder diagram or
instruction list, which control-program-development
supporting apparatus having,

5   an instruction counting unit which counts the
appearance frequency of each instruction used for the control
program;

a pattern-matching-table generating unit which
generates a pattern-matching table in which instructions
10   are listed starting with the highest appearance frequency
in accordance with results counted by the
instruction-counting unit; and

a compiler which compiles the control program into
directly executable codes by the programmable controller
15   while pattern-matching the instructions listed in the
pattern matching table in order.

(Note 5)   A programmable controller which performs
sequential processing in accordance with execution codes
generated by compiling a control program, said programmable
20   controller comprising:

a storing unit which stores the execution codes;

a microprocessor to be directly executed by the
execution codes,

wherein the execution codes include binary data
25   generated by compressing the control program; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development

5 supporting apparatus having,

a control-program dividing unit which divides the control program into a plurality of blocks; and

a compiler which compiles all or some of the blocks into execution codes directly executable by a programmable

10 controller.

(Note 6)   A programmable controller which performs sequential processing in accordance with execution codes generated by compiling a control program, said programmable controller comprising:

15 a storing unit which stores the execution codes;

a microprocessor to be directly executed by the execution codes,

wherein the execution codes include binary data generated by compressing the control program; and

20 a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

25 a control-program dividing unit which divides the

control program into a plurality of blocks;

a control-program converting unit which converts all or some of the blocks into advanced-language control programs described with a universal-computer-readable advanced language every blocks concerned; and

a compiler which compiles all or some of universal-computer-readable advanced programming languages corresponding every above block into directly executable codes by a programmable controller.

(Note 7) A programmable controller which performs sequential processing in accordance with execution codes generated by compiling a control program, said programmable controller comprising:

a storing unit which stores the execution codes;

a microprocessor to be directly executed by the execution codes,

wherein the execution codes include binary data generated by compressing the control program; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

a control-program converting unit which converts the control program into an advanced-programming-language

83

control   program   described   with   a
universal-computer-readable   advanced   programming
language;

a debugging-code generating unit which generates a debugging control program by inserting a line number concerned into a part corresponding to each line constituting the instruction list in source codes constituting the advanced-programming-language control program; and

a debugging executing unit which displays each line of the instruction list and the execution part of the advanced-programming-language control program by relating the former with the latter.

(Note 8)   A programmable controller which performs sequential processing in accordance with execution codes generated by compiling a control program, said programmable controller comprising:

a storing unit which stores the execution codes;

a microprocessor to be directly executed by the execution codes,

wherein the execution codes include binary data generated by compressing the control program; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction   list,   which   control-program-development

84

supporting apparatus having,

an instruction counting unit which counts the appearance frequency of each instruction used for the control program;

5   a pattern-matching-table generating unit which generates a pattern-matching table in which instructions are listed starting with the highest appearance frequency in accordance with results counted by the instruction-counting unit; and

10   a compiler which compiles the control program into directly executable codes by the programmable controller while pattern-matching the instructions listed in the pattern matching table in order.

(Note 9)   A programmable controller which performs 15   sequential processing in accordance with a control program described with a sequential-control language such as a ladder diagram or instruction list, said programmable controller comprising:

a storing unit which stores the control program;

20   an instruction counting unit which counts the appearance frequency of each instruction used for the control program;

a pattern-matching-table generating unit which generates a pattern-matching table in which instructions 25   are listed starting with the highest appearance frequency

in accordance with results counted by the instruction-counting unit;

an interpreting unit which executes the control program while pattern-matching the instructions listed in the pattern-matching table in order and interpreting the control program into directly-executable execution codes by the programmable controller; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

a control-program dividing unit which divides the control program into a plurality of blocks; and

a compiler which compiles all or some of the blocks into execution codes directly executable by a programmable controller.

(Note 10) A programmable controller which performs sequential processing in accordance with a control program described with a sequential-control language such as a ladder diagram or instruction list, said programmable controller comprising:

a storing unit which stores the control program;

an instruction counting unit which counts the appearance frequency of each instruction used for the control

86

program;

a pattern-matching-table generating unit which generates a pattern-matching table in which instructions are listed starting with the highest appearance frequency in accordance with results counted by the instruction-counting unit;

an interpreting unit which executes the control program while pattern-matching the instructions listed in the pattern-matching table in order and interpreting the control program into directly-executable execution codes by the programmable controller; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

a control-program dividing unit which divides the control program into a plurality of blocks;

a control-program converting unit which converts all or some of the blocks into advanced-language control programs described with a universal-computer-readable advanced language every blocks concerned; and

a compiler which compiles all or some of universal-computer-readable advanced programming languages corresponding every above block into directly

87

executable codes by a programmable controller.

(Note 11) A programmable controller which performs sequential processing in accordance with a control program described with a sequential-control language such as a ladder diagram or instruction list, said programmable controller comprising:

a storing unit which stores the control program;

an instruction counting unit which counts the appearance frequency of each instruction used for the control program;

a pattern-matching-table generating unit which generates a pattern-matching table in which instructions are listed starting with the highest appearance frequency in accordance with results counted by the instruction-counting unit;

an interpreting unit which executes the control program while pattern-matching the instructions listed in the pattern-matching table in order and interpreting the control program into directly-executable execution codes by the programmable controller; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

88

a control-program converting unit which converts the control program into an advanced-programming-language control program described with a universal-computer-readable advanced programming language;

a debugging-code generating unit which generates a debugging control program by inserting a line number concerned into a part corresponding to each line constituting the instruction list in source codes constituting the advanced-programming-language control program; and

a debugging executing unit which displays each line of the instruction list and the execution part of the advanced-programming-language control program by relating the former with the latter.

(Note 12) A programmable controller which performs sequential processing in accordance with a control program described with a sequential-control language such as a ladder diagram or instruction list, said programmable controller comprising:

a storing unit which stores the control program;

an instruction counting unit which counts the appearance frequency of each instruction used for the control program;

a pattern-matching-table generating unit which generates a pattern-matching table in which instructions

are listed starting with the highest appearance frequency in accordance with results counted by the instruction-counting unit;

an interpreting unit which executes the control program while pattern-matching the instructions listed in the pattern-matching table in order and interpreting the control program into directly-executable execution codes by the programmable controller; and

a control-program-development supporting apparatus that develops a control program described with a sequential-control language such as a ladder diagram or instruction list, which control-program-development supporting apparatus having,

an instruction counting unit which counts the appearance frequency of each instruction used for the control program;

a pattern-matching-table generating unit which generates a pattern-matching table in which instructions are listed starting with the highest appearance frequency in accordance with results counted by the instruction-counting unit; and

a compiler which compiles the control program into directly executable codes by the programmable controller while pattern-matching the instructions listed in the pattern matching table in order.

Although the invention has been described with respect to a specific embodiment for a complete and clear disclosure, the appended claims are not to be thus limited but are to be construed as embodying all modifications and alternative constructions that may occur to one skilled in the art which fairly fall within the basic teaching herein set forth.